Pure Dominance Constraints

Manuel Bodirsky[1] and Martin Kutz[2]

[1] Humboldt Universität zu Berlin, Germany
bodirsky@informatik.hu-berlin.de
[2] Freie Universität Berlin, Germany
kutz@math.fu-berlin.de

Abstract. We present an efficient algorithm that checks the satisfiability of *pure dominance constraints*, which is a tree description language contained in several constraint languages studied in computational linguistics. Pure dominance constraints partially describe unlabeled rooted trees. For arbitrary pairs of nodes they specify sets of admissible relative positions in a tree. The task is to find a tree structure satisfying these constraints.
Our algorithm constructs such a solution in time $\mathcal{O}(m^2)$ where m is the number of constraints. This solves an essential part of an open problem posed by Cornell.

Keywords. Efficient algorithms, tree descriptions, constraint satisfaction

1 Introduction

Tree description languages became an important tool in computational linguistics over the last twenty years. Grammar formalisms have been proposed which derive logical descriptions of trees representing the syntax of a string [18, 22, 8]. Acceptance of a string is then equivalent to the satisfiability of the corresponding logical formula.

In computational semantics, the paradigm of *underspecification* aims at manipulating the partial description of tree-structured semantic representations of a sentence rather than at manipulating the representations themselves [21, 9]. So the key issue in both, constraint based grammar and semantic formalisms, is to collect partial descriptions of trees and to *solve* them, i.e., to find a tree structure that satisfies all constraints.

Cornell [5] introduced a simple but powerful tree description language. It contains literals for dominance, precedence and equality between nodes, and disjunctive combinations of the these. He also gave a saturation algorithm based on local propagations but the completeness proof for the algorithm turned out to be wrong [6]. Recently Bodirsky, Duchier and Niehren [personal communication] in fact found a counterexample to the completeness of this algorithm.

In this paper, we will always talk about finite rooted unlabeled trees without any order on the children of a node. The constraint language that we are considering contains variables denoting the nodes in a tree together with dominance,

H. Alt and A. Ferreira (Eds.): STACS 2002, LNCS 2285, pp. 287–298, 2002.

disjointness, and equality constraints between them. Moreover, we can combine the above constraints disjunctively. For example, we can state that some node x must either lie above some other node y or must be disjoint to it, i.e., lie in a different branch of the tree. We call the resulting tree description language *pure dominance constraints*.

In contrast to pure dominance constraints, the constraint language introduced by Cornell is based on trees with an order on the children of the tree nodes. His *precedence constraints* are stronger than our disjointness constraints. They allow to state that some node x in the tree lies to the left of some other node y. Interestingly, the counterexample to the completeness of Cornell's algorithm does not make use of this distinction. It can be formulated using disjointness only instead of precedence.

Pure dominance constraints are a fragment of *dominance constraints* with set operators [18, 3, 7] which have applications both in syntax and in semantics of computational linguistics [22, 12, 9]. Those provide literals for *labeling*, *arity*, *parent-child relations*, and *dominance* to partially describe finite labeled trees. Checking satisfiability of dominance constraints is NP-complete [15]. For the restrictive, though linguistically still relevant fragment of *normal dominance constraints*, there is a polynomial time satisfiability algorithm [2, 14]. Dominance constraints with set operators have been studied in [7] and are important for formulating algorithms for extensions of dominance constraints with more expressive constraints [10, 4]. Pure dominance constraints are a strict subset of dominance constraints with set operators since they lack the possibility to specify labeling and arity of nodes in the described tree.

Hilfinger, Lawler and Rote investigated a similar problem. They gave an efficient algorithm that minimizes the depth of a given tree satisfying additional tree constraints [13], which has applications in compiling block structured computer languages.

If nodes in a tree are interpreted as intervals over the real line, pure dominance constraints translate into a fragment of Allen's interval algebra [1] where the intervals are a laminar family, i.e., they are overlap-free. Dominance between nodes then corresponds to containment of intervals. Allen's full interval constraint logic has many applications in temporal reasoning but is NP-complete in its unrestricted form. Nebel and Bürckert [20] presented an algorithm for the largest tractable subclass "ORD-Horn" of Allen's interval algebra, which decides satisfiability in time $\mathcal{O}(n^3)$ and constructs a solution in time $\mathcal{O}(n^5)$, where n is the number of intervals. Note that this algorithm cannot be used to solve pure dominance constraints since the translation into interval constraints may have non-laminar solutions, which do not retranslate into trees.

In this paper, we will present an efficient and complete algorithm that tests satisfiability of pure dominance constraints by directly constructing a solution to the problem instance. It runs in time $\mathcal{O}(m^2)$, where m is the number of constraints in the input. This is considerably faster than the known algorithms for the related problems on interval constraints. The performance is achieved by

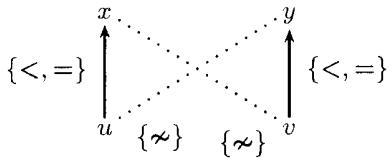

Fig. 1. A visualization of the unsatisfiable pure dominance constraint $\{\ x\{\nsim\}v, u\{<,=\}\ x, v\ \{<,=\}\ y, u\ \{\nsim\}\ y\ \}$.

a recursive greedy strategy that works directly on the constraint graph avoiding the consistent path technique and saturation algorithms of, for example, [17].

2 Pure Dominance Constraints

Pure dominance constraints are tree specifications. So let us first fix some conventions for trees. We will always consider finite rooted directed trees, the arcs pointing upward towards the root. Normally the nodes of a tree will be denoted by $a, b, c, \ldots$. We write $a \leq b$ and say that b dominates a if there is a path from a to b. We write $a < b$ for $a \leq b$ and $a \neq b$. The expression $a \sim b$ denotes comparability of a and b, that is, $a \leq b$ or $a \geq b$. Two incomparable nodes $a \nsim b$ are called *disjoint*. Note that for every pair a, b of nodes in a tree, exactly one of the following cases holds:

$$a < b, \quad a > b, \quad a \nsim b, \quad a = b.$$

Pure dominance constraints allow to partially describe the structure of a tree by use of arbitrary disjunctions of these four cases.

Definition 1. Let V be a set of variable symbols ranged over by $\{x, y, z, \ldots\}$, and $\mathcal{R} := \{<, >, \nsim, =\}$ a set of binary relation symbols. Then a *pure dominance constraint (PDC)* is a set Φ of literals of the form $x\ \Omega\ y$ where $\Omega \subseteq \mathcal{R}$ and $x, y \in V$. We write V_Φ for the variable symbols occurring in Φ.

A *Solution* (T, α) of a PDC Φ is a tree T together with a surjective mapping $\alpha : V_\Phi \to T$ from the used variables onto the nodes of this tree *satisfying* all literals in Φ. A literal $x\ \Omega\ y$ is *satisfied* by a solution (T, α) if $\alpha(x)\ \rho\ \alpha(y)$ holds in the tree T for some relation $\rho \in \Omega$.

We will omit the subscript of V_Φ whenever the reference to a specific pure dominance constraint Φ is clear. For readability we will often write $\overline{x}$ for $\alpha(x)$.

If a PDC has a solution we call it *satisfyable*. Note that we are not only interested in the *constraint satisfaction problem* for PDCs, that is, the question whether a given PDC is satisfiable.[1] We also want to *solve* a given constraint Φ efficiently, i.e., compute a solution (T, α).

[1] Note that the constraint satisfaction problem considered here does not fall into the class of *classical constraint satisfaction problems* since we do not map the variables into a *finite* template. Hence the classification of tractable constraint satisfaction problems of, e.g., [11] does not apply to our problem.

Figure 1 shows an unsatisfiable constraint. It has no solution since there is no candidate for a root and the mapping α from variables into the tree must be surjective.

The requirement that α be onto is natural since we are looking for a tree structure on V_Φ and not just any homomorphism into a tree. Nevertheless, dropping this restriction only changes the problem marginally. If (T, α) is a solution with nonsurjective α, we can remove every nonroot node in T that does not have a preimage, making all children of α children of α's parent. It remains the problem to map some $x \in V_\Phi$ to the root of T. As shown in Figure 1 this may not be possible although a nonsurjective solution might exist. But the general case can easily be transformed into the surjective case: Given a PDC Φ, we introduce a new variable x_0 and constraints $x_0 \{>\} x$ for each $x \in V_\Phi$. This forces x_0 to map to the root of T and therefore, finding an arbitrary solution for Φ is equivalent to finding a surjective one for the extended constraint.

3 Restricted Constraints

As a first step towards solving pure dominance constraints, we show that each PDC can be expressed with the following three types of literals only:

$$x \{<, =\} y, \quad x \{\nsim, =\} y, \quad x \{<, >, \nsim\} y. \tag{1}$$

The singletons $\{>\}$, $\{<\}$, $\{\nsim\}$ and $\{=\}$ can be written as intersections of these; and $\{>, =\}$ is simply the first literal of (1) flipped.

If we show how to express the literals

$$x \{<, >, =\} y \quad \text{and} \quad x \{<, \nsim, =\} y \tag{2}$$

with those in (1) we are finished, since $\{<, \nsim\}$, $\{>, \nsim\}$ and $\{<, >\}$ are representable as intersections of literals from (1) and (2). The two extremal sets $\{<, >, \nsim, =\}$ and $\emptyset$ are not needed since the former imposes no restrictions on the tree and the latter is, by definition, unsatisfyable. We can construct the literals in (2) from those in (1) by means of auxiliary variables:

For each literal $\phi = x \{<, >, =\} y$, we introduce a new variable z and replace ϕ by the two literals $z \{<, =\} x$ and $z \{<, =\} y$. By tree shape, these two imply $\overline{x} \sim \overline{y}$, just as ϕ does.

For each literal $\phi = x \{<, \nsim, =\} y$, we introduce a new variable z and replace ϕ by the two literals $x \{<, =\} z$ and $z \{\nsim, =\} y$. Now either $\overline{z} = \overline{y}$, which implies $\overline{x} \leq \overline{y}$, or $\overline{z} \not\sim \overline{y}$, and therefore $\overline{x} \not\sim \overline{y}$ by tree shape. Since the transformed constraint implies the original one, a solution (T, α) for the former is also a solution for the latter if we restrict α to the original variables.

The above modifications also maintain solvability. Let (T, α) be a solution for the original constraint Φ containing $x \{<, >, =\} y$. Then we have $\overline{x} \sim \overline{y}$. Assume wlog. that $\overline{x} \leq \overline{y}$. Letting $\overline{z} := \overline{x}$ then satisfies the new constraints $x \{>, =\} z$ and $z \{<, =\} y$. If Φ contained a literal $x \{<, \nsim, =\} y$, we can extend the solution (T, α) by $\overline{z} := \overline{y}$ if $\overline{x} < \overline{y}$, and $\overline{z} := \overline{x}$ otherwise. In both cases $x \{<, =\} z$ and $z \{\nsim, =\} y$ are satisfied in the modified constraint.

Thus we can express any PDC with the three basic literals from (1). We give them special names: The literal $x\{\nsim,=\}y$ is called *parallelity* literal—in analogy to a pair of parallel lines in the plane which are either disjoint or coincide. The literal $x\,\{<,>,\nsim\}\,y$ is called a *distinctness* and $x\,\{<,=\}\,y$ a *dominance* literal. Our algorithm works on PDCs containing these three kinds of literals only. Note that the above transformations may introduce a new node for each constraint in Φ. Hence, when dealing with running times, we will have to state whether we consider the general problem or the reduced one.

4 Preparations

We assume that our PDC has already been transformed so that it only contains dominance, parallelity and distinctness literals. We define the *dominance graph* $G = (V, E)$ of a PDC Φ on the variables $V := V_\Phi$ by letting $(x, y) \in E$ iff $x\,\{<,=\}\,y$ is in the PDC. Parallelity and distinctness literals are represented by two symmetric irreflexive binary relations P and D, respectively. We let $x\ P\ y$ iff $x\{\nsim,=\}y$ and $x\ D\ y$ iff $x\{<,>,\nsim\}y$. Because our algorithm and its correctness proofs are given in terms of graph theory, we will normally call elements $x \in V$ *nodes*. This should cause no confusion; it will always be clear whether we talk about nodes of V or T.

It turns out useful to define the relation $\leq$ on the set V, letting $x \leq y$ if and only if there is a directed path from x to y in the graph G. Obviously $\leq$ is a quasi order (that is, it is transitive, reflexive, but not necessarily antisymmetric) and we may use the symbols $\geq, <, >$ as usual. As with trees, we use the expression $x \sim y$, which is a short for $x \leq y$ or $x \geq y$. Observe that a binary relation $\leq$ is now defined on the tree T and on the variable set V. Again, the reference will always be clear.

The problem can now be restated as follows: Given an instance (V, E, P, D) of a pure dominance constraint, find a tree T and a surjection $\alpha : V \to T$ so that the following conditions are satisfied:

$x \leq y \Rightarrow \overline{x} \leq \overline{y}$	dominance
$x\ P\ y \Rightarrow \overline{x} \not\sim \overline{y}$ or $\overline{x} = \overline{y}$	parallelity
$x\ D\ y \Rightarrow \overline{x} \neq \overline{y}$	distinctness

The critical parts of the problem are pairs of parallel elements that may not become equivalent in the solution. If D is empty, we can simply map all nodes to the root. And it is also easy to see how to construct a solution if P is empty. So let us consider pairs of nodes that are mapped to incomparable nodes in the tree.

Lemma 1. *Let (T, α) be a solution to the instance (V, E, P, D). Consider any sequence $(a_0, a_1, a_2, \ldots, a_r)$ in V with $a_{i-1} \sim a_i$ for $1 \leq i \leq r$ and with $\overline{a_0} \not\sim \overline{a_r}$. Then there exists an index $j \in \{1, \ldots, r-1\}$ with $\overline{a_j} > \overline{a_0}$ and $\overline{a_j} > \overline{a_r}$.*

Proof. By induction on the length r. For $r = 1$ there is nothing to show since the combination $a_0 \sim a_1$, $a_0\ P\ a_1$, and $a_0\ D\ a_1$ has no solution. If $r = 2$, uncomparability of a_0 and a_2 directly implies $\overline{a_1} > \overline{a_0}, \overline{a_2}$.

So let $r > 2$. If $a_{i-1} \sim a_{i+1}$ for some $0 < i < r$, we may remove a_i from the sequence and our claim follows by induction. Otherwise, the relations in the sequence alternate and hence there exists an $i \in \{1, \ldots, r-1\}$ with $a_i < a_{i-1}$ and $a_i < a_{i+1}$. By tree shape, $\overline{a_{i-1}}$ and $\overline{a_{i+1}}$ have to be comparable. Assume wlog. $\overline{a_{i-1}} \leq \overline{a_{i+1}}$. If we let $E' := E \cup \{(a_{i-1}, a_{i+1})\}$ then (T, α) is also a solution to the instance (V, E', P, D). Now $(a_0, \ldots, a_{i-1}, a_{i+1}, \ldots, a_r)$ is a sequence of length $r-1$ with any two neighboring elements related in (V, E') and again the lemma follows by induction. □

As an example, let us consider the constraint graph in Figure 2. In any solution, we must have $\overline{x_1} \not\sim \overline{x_3}$. Thus by Lemma 1, we get $\overline{x_2} > \overline{x_1}, \overline{x_3}$. Now $\overline{x_2} \not\sim \overline{x_4}$ would contradict Lemma 1; hence $\overline{x_2} = \overline{x_4}$ by parallelity. The positions of u, v, and w are not of interest.

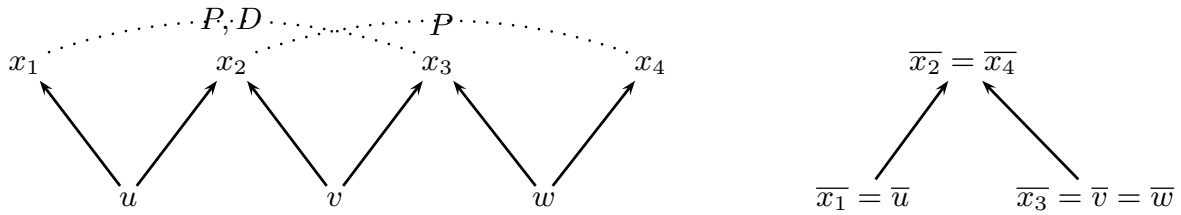

Fig. 2. A constraint graph and a solution

5 The Algorithm

Our algorithm is based on the recursive function `treeify`. Given a set $W \subseteq V$ of nodes, `treeify` creates a new tree T and defines the mapping α on W, yielding $\alpha(W) = T$. Finally the tree T is returned. More specifically, `treeify` first creates an empty tree T containing only a root r and maps a certain subset $S \subseteq W$ to r. Afterwards it calls itself recursively on subsets $W_i \subseteq W \setminus S$. The trees returned by these recursive calls are then linked to the root r.

Before we come to the crucial part of this function, namely the choice of the sets S and W_i, let us recollect some basic notions from graph theory: An *undirected* path in a directed graph may use arcs in any direction, ignoring their orientation. A strongly (weakly) connected component of G is a maximally induced subgraph U of G with a directed (an undirected) path from a to b for any two nodes $a, b \in U$. We introduce the expression $(V, E)|_W := (W, E \cap W^2)$ for the subgraph of (V, E) induced by a node set $W \subseteq V$.

So let us find out how to choose the set S. By construction we get $\overline{x} > \overline{y}$ for all $x \in S$ and $y \in W \setminus S$. Thus dominance demands that S be closed under

```
treeify(W):
    compute the strongly connected components of the graph (V, E ∪ P)|_W;
    if no free component exists then exit "problem has no solution"
    else pick a free component S;
    create new tree T with root r;
    for each x ∈ S do α[x] := r od;
    compute the weakly connected components W_1, ..., W_k of the graph (V, E)|_{W\S};
    for i = 1 to k do
        T_i := treeify(W_i);
        link T_i directly under r;
    od;
    return T;
```

Fig. 3. The function `treeify`

directed reachability in (V, E), i.e., no E-edges may leave S. Suppose now that a pair of parallel elements $a\ P\ b$ gets split by S, say $a \in S$ but $b \notin S$. This implies $\overline{a} > \overline{b}$, in contradiction to parallelity. Hence either $a, b \in S$ or $a, b \notin S$ for $a\ P\ b$. And finally, S may obviously not contain a pair of distinct elements $u\ D\ v$. These observations lead to the following definition.

Definition 2. A strongly connected component C of the graph $(V, E \cup P)|_W$, $W \subseteq V$, is called a *terminal* component (of W) if its outdegree in $(V, E \cup P)|_W$ is zero. That is, $(E \cup P) \cap (C \times (W \setminus C)) = \emptyset$. We call a terminal component $C \subseteq W$ *free* if $x \not D\ y$ for all $x, y \in C$.

Note that the graph $(V, E \cup P)|_W$ contains—in addition to the arcs from E—a bidirectional edge between each pair $x\ P\ y$ of parallel nodes in W.

For the set S, the function `treeify` picks an arbitrary free component of W. We will show later that, given the whole problem has a solution, such a free component always exists. It remains to choose the partition $W_1, \ldots, W_k$ of $W \setminus S$. By construction, elements in different sets W_i of this partition will become incomparable in the tree. So we may simply choose the W_i to be the weakly connected components of the graph $(V, E)|_{W \setminus S}$. This way parallelity and distinctness constraints between different components are automaticly resolved. The proofs will be given below.

Figure 3 gives a detailed description of the function `treeify`. The whole algorithm for an input (V, E, P, D) just consists of the function call `treeify`(V). Note that although the tree structure T is created recursively, the mapping α is not passed at function calls but is accessed globally. We assume that at the beginning we have $\alpha(x) =$ `undef` for all $x \in V$.

5.1 Correctness Proof

Soundness. We first show that the algorithm only returns correct solutions, i.e., it satisfies the conditions dominance, parallelity and distinctness of Section 4.

To verify dominance, we need to show that reachability in the induced graphs $(V,E)|_W$ is equivalent to reachability in the input graph (V,E), which defines the relation $\leq$ on V.

Lemma 2. *Let $W \subseteq V$ be a node set that occurs as an argument to the function* `treeify` *throughout the execution of the algorithm. For any pair $x, y \in W$, each directed path p in (V,E) from x to y is also a path in $(V,E)|_W$.*

Proof. By induction over the recursion. For the initial case $W = V$ there is nothing to do. So assume that the statement is true for some call `treeify`(W), and that we have just chosen a free component S. We show that it remains true for all recursive calls of `treeify` on weakly connected components W' of $(V,E)|_{W \setminus S}$. Let x, y be nodes in W'. No path p from x to y contains a node from S since S is terminal. Hence, p lies completely in W' because W' contains all nodes $z \in W \setminus S$ that are reachable from x. □

Now dominance follows easily by induction. Consider a call `treeify`(W) and a pair $x, y \in W$ with $x \leq y$. If $x \in S$, we also have $y \in S$ and thus $\overline{x} = \overline{y}$. If $x \notin S$ and $y \in S$, we get $\overline{y} = r$ while x is passed on through a recursive call and hence $\overline{x}$ will lie in a subtree below r. Otherwise, x and y both fall into the same weakly connected component W_i and we may conclude dominance by induction.

To verify parallelity, we again consider an arbitrary call `treeify`(W). Let x, y be nodes in W. If $x, y \in S$ we have $\overline{x} = \overline{y}$. Otherwise, none of x and y lies in S since these nodes are strongly connected in $(V, E \cup P)|_W$. If they lie in different weakly connected components of $(V,E)|_{W \setminus S}$, their images $\overline{x}$ and $\overline{y}$ will fall into different subtrees of r. Hence $\overline{x} \not\sim \overline{y}$. If they lie in the same component, our statement again follows by induction.

Condition distinctness is satisfied since any two nodes x, y with $\overline{x} = \overline{y}$ lie in the same free component S of some instance of `treeify`. But since S is free such pairs satisfy $x \not\mathrel{D} y$.

The constructed mapping $\alpha : V \to T$ is obviously a surjection.

Completeness. So we know that the algorithm only returns correct solutions. We now show that it also always finds one if the PDC has a solution. To this end, we assume that the input (V,E,P,D) has a solution (T,α). Then we consider an arbitrary instance of `treeify`(W) that is executed by the algorithm and prove that the set W must contain a free component.

The basic idea for the proof is to construct iteratively a sequence $x_0, x_1, x_2, \ldots$ in W with $\overline{x_0} \leq \overline{x_1} \leq \ldots$. Some of these inequalities will be strict. For the others with $\overline{x_i} = \overline{x_{i+1}}$ we will guaranty the existence of a path from x_i to x_{i-1} in $(V, E \cup P)|_W$. From these paths we will be able to conclude that our sequence eventually enters a free component. For preparation we prove the following extension of Lemma 1.

Lemma 3. *Let (T,α) be a solution to the input (V,E,P,D) and let W be the argument of a call to* `treeify`*. Then for any pair $u, v \in W$ with $\overline{u} \not\sim \overline{v}$, there exists a node $y \in W$ with $\overline{y} > \overline{u}, \overline{v}$.*

Proof. We distinguish two cases. If $W = V$, α maps neither u nor v to the root r of the tree T since otherwise $\overline{u} \sim \overline{v}$. Hence any $y \in \alpha^{-1}(r)$ has the desired property.

If $W \neq V$, the graph $(V, E)|_W$ is weakly connected. So there is an undirected path q in this graph connecting u and v. By Lemma 1 there exists a node $y \in W$ with $\overline{y} > \overline{u}$ and $\overline{y} > \overline{v}$. □

The next lemma is the basis for our construction of the sequence $x_0, x_1, x_2, \ldots$.

Lemma 4. *Let (T, α) be a solution to the input (V, E, P, D) and let W be the argument of a call to* `treeify`*. Let C be a strongly connected component of $(V, E \cup P)|_W$ that is not free and let $x \in C$. Then there either exists*

1. *a node $y \in W$ with $\overline{y} > \overline{x}$ or*
2. *a node $w \in W \setminus C$ satisfying $\overline{w} \geq \overline{x}$, together with a directed path q in $(V, E \cup P)|_W$ from x to w.*

Proof. If C is terminal, there exist two nodes $a, b \in C$ with $a\ D\ b$ since C is not free. Choose two directed paths $p_1 = (x = u_1, \ldots, u_k = a)$ and $p_2 = (a = u_k, \ldots, u_l = b)$ in $(V, E \cup P)|_C$. By distinctness, we have $\overline{u_k} \neq \overline{u_l}$. Hence there must be an index $j < l$ with $\overline{x} = \overline{u_j} \neq \overline{u_{j+1}}$.

Since $(u_j, u_{j+1}) \in E \cup P$, we have $u_j \leq u_{j+1}$ or $u_j\ P\ u_{j+1}$. In the former case let $y := u_{j+1}$; then $\overline{y} > \overline{u_j} = \overline{x}$. In the latter case we get $\overline{u_j} \not\sim \overline{u_{j+1}}$ and Lemma 3 yields the desired $y \in W$.

If C is not terminal there exist two nodes $z \in C$ and $w \in W \setminus C$ with $(z, w) \in E \cup P$. Since $(z, w) \in P$ would imply $w \in C$, we even have $z \leq w$. Let $p = (x = u_0, \ldots, u_k = z)$ be a directed path in $(V, E \cup P)|_C$. Consider the index set $I = \{i \mid 0 \leq i < k,\ \overline{u_i} \not\leq \overline{u_{i+1}}\}$.

If $I \neq \emptyset$, let $j := \min I$. We have $u_j \not\leq u_{j+1}$ by dominance and thus $u_j\ P\ u_{j+1}$ by the definition of p. Therefore $\overline{x} \leq \overline{u_j} \not\leq \overline{u_{j+1}}$; and by parallelity even $\overline{u_j} \not\sim \overline{u_{j+1}}$. Now Lemma 3 again yields the sought-after $y \in W$.

If $I = \emptyset$, we have $\overline{x} \leq \overline{z} \leq \overline{w}$ and the path $q = (u_0, \ldots, u_k, w)$ connects x and w in $(V, E \cup P)|_W$. □

Theorem 1. *If the instance (V, E, P, D) has a solution then for each function call* `treeify`*(W) throughout the execution of the algorithm, the respective graph $(V, E \cup P)|_W$ contains a free component.*

Proof. We pick an arbitrary node $x_0 \in W$. If it lies in a free component we are finished. If not, we apply Lemma 4 to x_0 yielding some $x_1 \in W$ with $\overline{x_1} > \overline{x_0}$ or only $\overline{x_1} \geq \overline{x_0}$. But in the latter case, x_1 is reachable from x_0 via a directed path in $(V, E \cup P)|_W$. We may repeat this step as long as no free component is found. This yields a sequence $x_0, x_1, x_2, \ldots$ in W.

We claim that this sequence is finite, i.e., some x_i will eventually lie in a free component. Assume for contradiction that it is infinite. Then we have $x_k = x_l$ for some pair of indices $k < l$. Since $\overline{x_{i+1}} > \overline{x_i}$ for any index $i \in \{k, \ldots, l-1\}$ would

imply the contradiction $\overline{x_k} < \overline{x_k}$, all pairs x_i, x_{i+1}, $k \leq i < l$ of neighbouring nodes are connected by a directed path. But then $x_k, x_{k+1}, \ldots, x_{l-1}$ all lie in the same strongly connected component of $(V, E \cup P)_W$, in contradiction to Lemma 4. □

5.2 Running Time

We measure the running time of our algorithm in terms of the number of nodes $n := \#V$ and constraints $m := \#E + \#P + \#D$. For our estimates, we assume $m \geq n - 1$. Otherwise, the graph $(V, E \cup P \cup D)$ is not connected and we can simply treat its weakly connected components independently and then link the resulting trees in any arbitrary order.

Theorem 2. *The algorithm runs in time* $\mathcal{O}(nm)$.

Proof. Computing the strongly connected components of $(V, E \cup P)|_W$ takes $\mathcal{O}(m)$ steps [16, Sec. 3]. Checking each components outdegree and internal distinctness constraints also takes $\mathcal{O}(m)$ steps. Creating the tree and linking the subtrees can be done in $\mathcal{O}(n)$ steps. Hence, each instance of `treeify` performs $\mathcal{O}(m)$ operations, not counting the time needed for the recursive calls.

Since in each instance of `treeify` at least one node is mapped to the tree, there happen no more than n calls to `treeify` throughout the whole execution of the algorithm. □

Observe that this result applies to the reduced constraint set only. Since the reduction of pure dominance constraints to their restricted form might introduce a new vertex for each constraint, constructing a solution for a general pure dominance constraint might take time quadratic in the number of constraints.

Note that we cannot improve upon this bound of Theorem 2 by arguing that the sets $W \subseteq V$ in calls to `treeify` become smaller and smaller throughout the recursion. As a counterexample, consider a dominance graph that consists of a single directed path with all neighbouring nodes distinct. Then in each instance `treeify`(W), the free component S contains only a single element and the remaining graph $(V, E)|_{W \setminus S}$ remains weakly connected. On this input, our algorithm actually performs $\Omega(nm)$ steps.

Possible improvements. We want to hint at a part in our algorithm that appears inefficient and which might be subject to future improvements. The strongly connected components of subgraphs of $(V, E \cup P)$ and the weakly connected components of subgraphs of (V, E) are computed over and over again. The algorithm does not reuse any connectivity information.

Mapping the component S to the tree in any instance of `treeify` only removes nodes from the respective graph, and splitting the remaining set $W \setminus S$ into its weakly connected components only deletes edges. Thus we never introduce nodes or edges. Therefore, it might be possible to reuse information by means of the decremental dynamic connectivity techniques presented in [23]. But those results only apply to undirected graphs and it is not clear how to extend them to

strongly connected components. And even if we could speed up the connectivity computations, we would still have to find the free components.

6 Conclusion

We introduced the tree description language *pure dominance constraints* which forms a natural fragment of constraint languages used in computational linguistics. It is expressive enough to contain a counterexample to the completeness of Cornell's algorithm.

We presented an algorithm that constructs a solution of pure dominance constraints in quadratic time. It is mainly based on weak and strong graph connectivity. We think that the running time can still be improved using decremental dynamic connectivity techniques.

A main contribution of this paper is the method of using greedy algorithms and graph theoretic concepts for constraint satisfaction problems instead of the consistent path method and saturation algorithms. We may ask if it is possible to avoid the consistent path method for Nebel and Bürckert's maximal tractable subclass of Allen's interval algebra, too. A faster algorithm for this subclass would be of practical importance since it could be used to bound the branching in backtracking algorithms for the important full interval algebra [19, 24]. A first step in this direction would be the extension of our algorithm such that it can also deal with precedence constraints as defined in the original language by Cornell.

Acknowledgements. We want to thank Denys Duchier and Joachim Niehren for pointing us to the constraint problem, and an anonymous referee for his usefull remarks.

References

1. J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.
2. E. Althaus, D. Duchier, A. Koller, K. Mehlhorn, J. Niehren, and S. Thiel. An efficient algorithm for the configuration problem of dominance graphs. In *Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms*, pages 815–824, Washington, DC, Jan. 2001.
3. R. Backofen, J. Rogers, and K. Vijay-Shanker. A first-order axiomatization of the theory of finite trees. *Journal of Logic, Language, and Information*, 4:5–39, 1995.
4. M. Bodirsky, K. Erk, A. Koller, and J. Niehren. Beta Reduction Constraints. In *Proceedings of the 12th International Conference on Rewriting Techniques and Applications*, Utrecht, 2001.
5. T. Cornell. On determining the consistency of partial descriptions of trees. In *32nd Annual Meeting of the Association for Computational Linguistics*, pages 163–170, 1994.
6. T. Cornell. On determining the consistency of partial descriptions of trees. `http://tcl.sfs.nphil.uni-tuebingen.de/~cornell/mss.html`, 1996.

7. D. Duchier and J. Niehren. Dominance constraints with set operators. In *First International Conference on Computational Logic (CL2000)*, LNCS, July 2000.
8. D. Duchier and S. Thater. Parsing with tree descriptions: a constraint-based approach. In *Sixth International Workshop on Natural Language Understanding and Logic Programming (NLULP'99)*, pages 17–32, Las Cruces, New Mexico, Dec. 1999.
9. M. Egg, A. Koller, and J. Niehren. The Constraint Language for Lambda Structures. *Journal of Logic, Language, and Information*, 2001. To appear.
10. K. Erk, A. Koller, and J. Niehren. Processing underspecified semantic representations in the constraint language for lambda structures. *Journal of Language and Computation*, 2001. To appear.
11. T. Feder and M. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: a study through datalog and group theory. *SIAM J. Comput.*, (28):57–104, 1999.
12. C. Gardent and B. Webber. Describing discourse semantics. In *Proceedings of the 4th TAG+ Workshop*, Philadelphia, 1998. University of Pennsylvania.
13. P. Hilfinger, E. L. Lawler, and G. Rote. Flattening a rooted tree. *Applied Geometry and Discrete Mathematics*, 4:335–340, 1991.
14. A. Koller, K. Mehlhorn, and J. Niehren. A polynomial-time fragment of dominance constraints. In *Proceedings of the 38th Annual Meeting of the Association of Computational Linguistics*, Hong Kong, Oct. 2000.
15. A. Koller, J. Niehren, and R. Treinen. Dominance constraints: Algorithms and complexity. In *Third International Conference on Logical Aspects of Computational Linguistics (LACL '98)*, Grenoble, France, Dec. 1998.
16. T. Lengauer. *Combinatorial algorithms for integrated circuit layout.* Wiley – Teubner, 1990.
17. A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.
18. M. P. Marcus, D. Hindle, and M. M. Fleck. D-theory: Talking about talking about trees. In *Proceedings of the 21st ACL*, pages 129–136, 1983.
19. B. Nebel. Solving hard qualitative temporal reasoning problems: Evaluating the efficiency of using the ORD-Horn class. 1(3):175–190, 1997.
20. B. Nebel and H.-J. Bürckert. Reasoning about temporal relations: A maximal tractable subclass of Allen's interval algebra. *Journal of the ACM*, 42(1):43–66, 1995.
21. M. Pinkal. Radical Underspecification. In *Proc. 10th Amsterdam Colloquium*, 1996.
22. J. Rogers and V. Shanker. Reasoning with descriptions of trees. In *Proceedings of the 30th Meeting of the Association for Computational Linguistics*, 1992.
23. M. Thorup. Decremental dynamic connectivity. *J. Algorithms*, 33(2):229–243, Nov. 1999.
24. P. van Beek. Exact and approximate reasoning about temporal relations. *Computational Intelligence*, 6:132–144, 1990.